Stability Analysis and Performance Comparison of Three 6to4 Relay Implementations

Sándor Répás, Viktor Horváth, and Gábor Lencse

***Abstract*— During the IPv6 deployment there is a frequently occurring situation where two IPv6 enabled hosts need to communicate with each other over a network that supports only IPv4. Application of the 6to4 IPv6 transition method can solve this problem. The performance and stability of the different 6to4 relay implementations is a very important subject. We measured the performance and tested the stability of three open source 6to4 relay implementations under Debian Linux, OpenBSD and OpenWRT platforms. We present and discuss our results, analyze the stability of the 6to4 relay implementations and compare their performance metrics. Our measurements methods may be useful for other researchers, and our results may help the system architects to choose the appropriate solution.**

***Keywords*—6to4 relay, IPv6 transition, network communication, performance evaluation, stability analysis**

I. Introduction

FOR more than two decades it is a known fact, that the size of the IPv4 address space is insufficient [1]-[2]. The lack of the IP addresses withholds the spread of the Internet and causes social and economic damage.

To prevent the IP address exhaustion, a new version of the Internet Protocol, the IPv6 has been developed. IPv6 was standardized in 1998 and published in the RFC 2460 [3], but it has not been widespread adopted. According to the statistics, less than 5.5% of the total amount of the traffic reached the Google servers used IPv6 protocol in December 2014 [4]. Three of the five Regional Internet Registries (RIR) already run out of their IPv4 address spaces [5]. The five RIRs have only 5.2 /8 ranges in total, whereas the IANA does not have more address space to assign to the five RIRs since 3 February 2011 [6]. The RIRs work according to strict policies and for a service provider it is a harder task than ever to get IPv4 address spaces. The speed up of the transition to the new protocol is inevitable. Several IPv6 transition techniques have been developed, which can help the process in different phases of the adoption of the new protocol on the Internet.

There are different situations to solve during the coexistence of the two versions of the IP protocol in the different phases of the transition process:

In theory, the best solution is the Dual Stack (DS) transition method [7], but with the requirements that the two communicating hosts and the network between them have to support a common version of the IP protocol, and because of the IPv4 exhaustion, there is not enough IPv4 addresses to use this solution. Thus, even though it could have been the best solution, now it is too late for using DS as an IPv6 transition method.

In a situation where an IPv6 only client computer needs to communicate with an IPv4 only server the DNS64 [8] and NAT64 [9] combination is a good solution. The performance, the stability and the application compatibility of some open source implementations of DNS64/NAT64 are examined and proved in [10]-[12].

If two IPv6 enabled hosts need to communicate with each other over an IPv4 network, they can use different tunneling methods. The 6in4 (also called manual tunnel) [13] with tunnel brokers [14]-[15], 6rd [16], Teredo [17] ISATAP [18] and 6to4 [19] have different requirements, benefits and drawbacks.

The above list is not exhaustive and a good survey of the different transition techniques can be found in [20].

The remainder of this paper is organized as follows: first, some properties of the 6to4 transition technique are introduced, second, a short survey of the results of the most current publications is given, third, the selected 6to4 relay implementations are introduced, fourth, our test environment is described, fifth, the performance measurement method of the different implementations is detailed, sixth, the results are presented and discussed, seventh, the comparison of our results is presented, finally, our conclusions are given.

II. The 6to4 Transition Technique

The 6to4 transition technique uses automatic tunnels, encapsulates the IPv6 packets into IPv4 packets [19] (using protocol number 41, as the configured IPv6 over IPv4 tunnel [21]). The main advantage of the automatic tunneling is the unnecessity of the manual configuration of the endpoint address of the tunnel. Automatic IPv6-over-IPv4 tunneling determines the IPv4 tunnel endpoint address from the IPv4 address embedded in the destination address of the IPv6 packet being tunneled. 6to4 protocol uses the reserved 2002::/16 6to4 prefix [22] to determine if a 6to4 tunnel creation is necessary. A 6to4 address is an IPv6 address constructed using a 6to4 prefix. The first 16 bits of the 6to4 address contain the 2002 hexadecimal value, whereas the next 32 bits contain the IPv4 address of the 6to4 tunnel endpoint. The next 16 bits can be used to create subnets, and the final 64 bits of the 6to4 address contain the interface ID.

A 6to4 router is an IPv6 router supporting a 6to4 pseudo-interface. It is normally the border router between an IPv6 site

Manuscript received February 20, 2015.

S. Répás is with the Széchenyi István University, Győr, 9026 Hungary (phone: 36-30-459-9292; e-mail: repas.sandor@sze.hu).

V. Horváth was with the Széchenyi István University, Győr, 9026 Hungary (e-mail: vhorvath@biztributor.hu).

G. Lencse is with the Széchenyi István University, Győr, 9026 Hungary (E-mail: lencse@sze.hu).

and a wide-area IPv4 network, whereas the 6to4 pseudo-interface is the point of the encapsulation of the IPv6 packets in the IPv4 packets (with other words: the tunnel end-point) [19]. If a 6to4 host have to communicate with a non 6to4 host (for example: native IPv6, Teredo) it needs to use a 6to4 relay router.

Several operating systems can work as a 6to4 router or 6to4 relay router, but for the correct operation, the 6to4 routers and relay routers need public IPv4 addresses.

A 6to4 relay router can be private or public. Public 6to4 relays use the 192.88.99.1 anycast address [23] from the 192.88.99.0/24 6to4 Relay anycast address range [24]. An estimation of the 6to4 relay routers published in 2006 [25]. According to the publication, 8 autonomous systems (AS-es) advertised the 192.88.99.0/24, whereas 6 AS-es advertised the 2002::/16 networks. At the end of the year 2014 these values were 14 and 11, according to the RIPEstat database [26].

It is a good practice, if the Internet Service Provider (ISP) provides a 6to4 relay for its customers in addition to other transition solutions. In this case the relay does not have to be public, and it can use the well-known anycast address, or a network specific address.

Though some security weaknesses are known of the 6to4 transition technique [27], it helps the implementation of the IPv6 protocol without the cooperation of the ISP.

More details of the operation of the 6to4 technique can be found in the publication [28], and in the related RFCs ([19], [24] and [27].

III. TESTED IMPLEMENTATIONS

The following widely used open source [29] (also called free software [30]) operating systems and their 6to4 implementations were chosen for the tests: Debian Linux sit, FreeBSD stf interface, OpenWRT 6to4 plus kmod-sit packages. The open source software can be freely used by anyone, and their licenses allow the performance benchmarks. These two arguments were the most important ones in our selection of the implementations for testing.

The following software versions were used:

- Debian 7.1.0_x86 – sit (obsolete)
- OpenWRT (Attitude Adjustment) 12.09_x86 – sit
- FreeBSD 9.1_x86 – stf.

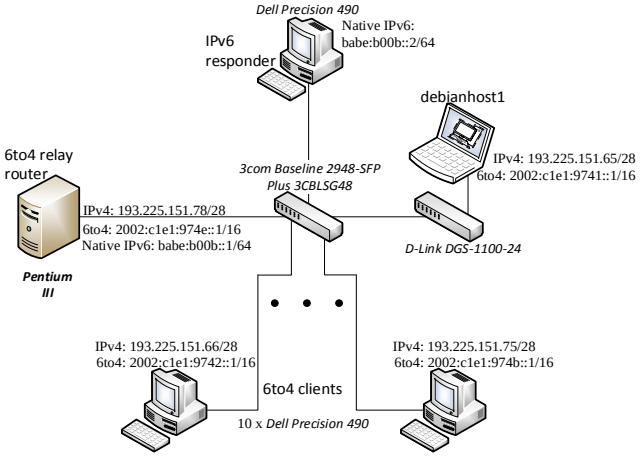

Fig. 1. Topology of the test network

IV. TEST ENVIRONMENT

A. Topology of the Network

An isolated test network was built for the performance and the stability measurements. The topology of the network can be seen in Fig. 1. Due to the isolation, any IPv4 and IPv6 addresses could be used on the network. The computer on the top of the figure played the role of the "internet" and responded all of the queries. The queries were generated by the 10 client computers which can be seen on the bottom of the figure. These computers played the role of the large number of the clients. The clients sent their queries by 6to4 through the 6to4 relay router to the "internet" computer. These queries were generated different levels of load on the 6to4 relay computer during the measurement process. The load was tuned by the number of the active clients. The laptop and the connecting switch on the right side of the figure were used for the control of the experiments.

B. Hardware Configurations

1000Base-TX connections were used on all of the network segments.

A specially low performance computer was built for the 6to4 relay computer so that the client computers could produce high enough load for overloading it. The main goal of the measurements was the comparison of the different implementations and not any hardware related investigation.

The 6to4 relay computer had an Intel D815EE2U motherboard, an Intel Pentium III (800MHz) processor, 128MB (100MHz) SDRAM and two TP-LINK TG-3269 REV 3.0 Gigabit PCI Ethernet NICs.

All of the ten clients and the responder computer were Dell Precision 490 workstations with same configuration: DELL 0GU083 motherboard with Intel 5000X chip-set, two Intel Xeon 5140 2.33GHz dual core processors (in the responder: Intel Xeon 5160 3GHz), 4x1GB 533MHz DDR2 SDRAM (accessed quad channel) and Broadcom NetXtreme BCM5752 Gigabit Ethernet controller (PCI Express).

C. Software configurations

Debian Linux 6.0.7 with 2.6.32-5-amd64 kernel and OpenBSD 5.3 64 bit version were installed on the clients, and the responder, respectively.

On the responder, NAT66 was used to simulate servers with different IPv6 addresses. The packets of the clients were redirected the computer itself, so that it can reply to them.

All of the client computers used sit interfaces and they used the network settings shown in Fig 1.

V. MEASUREMENT METHOD

The load was generated by `ping6` commands with the following Bash shell script:

```
#!/bin/bash
i=`cat /etc/hostname | grep -o '[0-9]'`
for b in {0..255}
do
  rm -rf $b
  mkdir $b
  for c in {0..252..4}
  do
```

```
    ping6 2001:738:2c01:8000::193.$i.$b.$c \
      -c8 -i0 >> $b/6to4-193-$i-$b-$c &
    ping6 2001:738:2c01:8000::193.$i.$b.$c \
      -c8 -i0 >> $b/6to4-193-$i-$b-$c &
    ping6 2001:738:2c01:8000::193.$i.$b.$((c+1)) \
      -c8 -i0 >> $b/6to4-193-$i-$b-$((c+1)) &
    ping6 2001:738:2c01:8000::193.$i.$b.$((c+1)) \
      -c8 -i0 >> $b/6to4-193-$i-$b-$((c+1)) &
    ping6 2001:738:2c01:8000::193.$i.$b.$((c+2)) \
      -c8 -i0 >> $b/6to4-193-$i-$b-$((c+2)) &
    ping6 2001:738:2c01:8000::193.$i.$b.$((c+2)) \
      -c8 -i0 >> $b/6to4-193-$i-$b-$((c+2)) &
    ping6 2001:738:2c01:8000::193.$i.$b.$((c+3)) \
      -c8 -i0 >> $b/6to4-193-$i-$b-$((c+3)) &
    ping6 2001:738:2c01:8000::193.$i.$b.$((c+3)) \
      -c8 -i0 >> $b/6to4-193-$i-$b-$((c+3))
  done
done
```

During the preliminary measurements the script was tuned to generate about 100% load on the CPU of the 6to4 relay computer with 10 clients.

The variable `i` contains the serial number of the actual client. The script contains two nested for cycles. The outer cycle with variable b from 0 to 255 runs 256 times, while the inner cycle with variable c from 0 to 252 (with stepping interval 4) runs 64 times. The core of the script contains 4 pairs of concurrent ping6 commands. Each of them sends out 8 ICMPv6 echo requests with almost zero time interval, in parallel, whereas the first 7 of them are started asynchronously with the `&` parameter. The last ping6 command at the end of the cycle is started normally thus the cycle waits for the execution of it. In a measurement, one client sends out 256*64*8*8= 1048576 ICMP echo requests in total to 256*64*4= 65536 different IP addresses.

In the series of measurements, the number of the clients was increased from one to ten. On the 6to4 relay computer, the `vmstat` command was used to log the CPU and memory consumption. For the proper operation of the `vmstat`, -10 nice value was used.

VI. Measurement Results

The results are presented in similar tables for all the tested 6to4 implementations. A detailed explanation is given for the first table only – the others are to be interpreted in the same way.

A. Debian 7.1.0_x86 – sit

The results have been listed in Table I. The first row shows the number of clients that executed the test script at the same time. The generated load on the 6to4 relay was proportional with the number of the clients. The second row contains the packet loss ratio. Rows 3, 4 and 5 show the average, the standard deviation and the maximum value of the response time, respectively. The average and the standard deviation of the CPU utilization of the 6to4 relay computer are shown in the rows 6 and 7. Row 8 contains the memory consumption of the 6to4 process on the relay computer. (This parameter can be measured with high uncertainty, because its value is very low and other processes than the 6to4 relay implementation may also influence the size of the used memory of the computer.) The last row shows the number of forwarded packets per seconds.

The graphical representation of the forwarded packets per second and the CPU utilization are shown in Fig. 2.

TABLE I. DEBIAN LINUX – SIT 6TO4 RELAY PERFORMANCE RESULTS

Number of clients		1	2	3	4	5	6	7	8	9	10
Packet loss (%)		0.002	0.006	0.008	0.013	0.020	0.035	0.035	0.037	0.048	0.061
Response time (ms)	Average	0.287	0.353	0.445	0.566	0.710	0.868	1.043	1.209	1.411	1.626
	Std. dev.	0.174	0.248	0.353	0.423	0.509	0.588	0.685	0.722	0.832	0.864
	Maximum	27.900	28.400	28.500	28.900	29.400	30.700	31.100	34.100	32.800	39.600
CPU Utilization (%)	Average	1.756	4.821	12.933	31.243	52.964	69.049	81.319	88.941	93.206	96.132
	Std. dev.	1.944	2.811	5.619	12.215	16.379	16.493	12.690	9.817	5.289	7.388
Memory consumption (kB)		10.855	10.418	10.363	10.594	10.824	10.996	10.855	10.994	10.828	11.137
Traffic volume (packets/sec)		18051	33953	46856	56534	62853	66947	69663	72304	73129	73050

TABLE II. OPENWRT (ATTITUDE ADJUSTMENT) 12.09_X86 – SIT 6TO4 RELAY PERFORMANCE RESULTS

Number of clients		1	2	3	4	5	6	7	8	9	10
Packet loss (%)		0.004	0.006	0.007	0.013	0.018	0.026	0.036	0.064	0.079	0.089
Response time (ms)	Average	0.314	0.402	0.568	0.733	0.909	1.118	1.358	1.616	1.873	2.160
	Std. dev.	0.161	0.239	0.330	0.420	0.508	0.583	0.652	0.705	0.773	0.829
	Maximum	25.000	25.300	25.500	25.500	26.500	27.100	27.000	27.100	27.300	28.100
CPU Utilization (%)	Average	10.067	45.015	70.713	87.188	94.979	97.540	98.467	98.916	99.066	99.288
	Std. dev.	3.188	5.593	5.828	9.376	7.954	7.462	4.991	4.567	4.824	4.410
Memory consumption (kB)		10.316	10.414	10.359	10.727	10.469	10.324	10.746	10.492	10.066	10.469
Traffic volume (packets/sec)		17595	32488	41906	49270	54196	56920	58272	58928	59332	58763

TABLE III. FREEBSD 9.1_X86 – STF 6TO4 RELAY PERFORMANCE RESULTS

Number of clients		1	2	3	4	5	6	7	8	9	10
Packet loss (%)		0.013	0.008	0.010	0.012	0.013	0.015	0.017	0.018	0.019	0.019
Response time (ms)	Average	0.315	0.456	0.681	0.941	1.268	1.637	2.011	2.385	2.740	3.126
	Std. dev.	0.111	0.171	0.314	0.404	0.450	0.457	0.463	0.466	0.480	0.490
	Maximum	22.200	9.220	12.800	15.400	17.600	18.100	18.800	18.500	19.600	19.400
CPU Utilization (%)	Average	51.525	77.110	88.994	96.380	98.482	99.435	99.395	99.371	99.462	99.859
	Std. dev.	6.899	5.140	6.465	7.398	7.593	3.447	5.336	6.445	5.971	0.475
Memory consumption (kB)		0.008	0.012	0.012	0.273	0.395	0.398	0.445	0.406	0.500	0.492
Traffic volume (packets/sec)		17594	30656	37613	41982	43681	43892	43875	43819	43970	43737

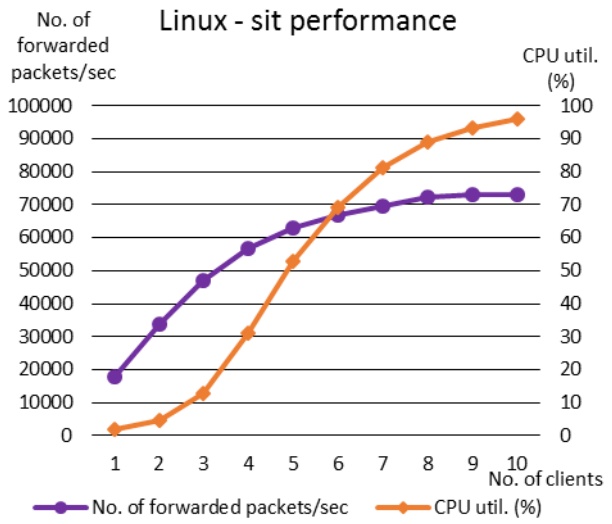

Fig. 2. Linux sit forwarded packets and CPU utilization

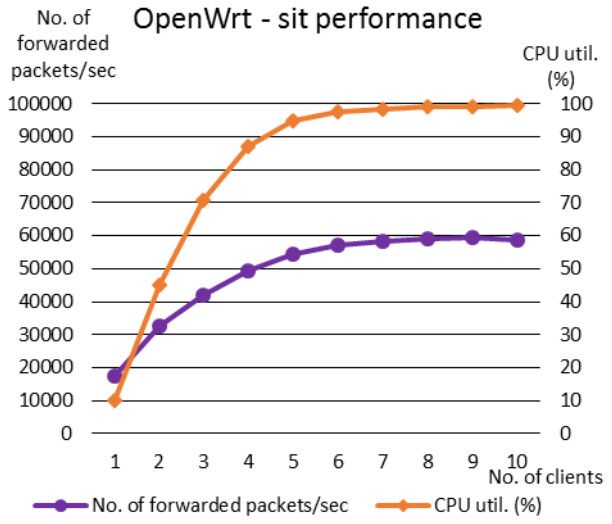

Fig. 3. OpenWrt sit forwarded packets and CPU utilization

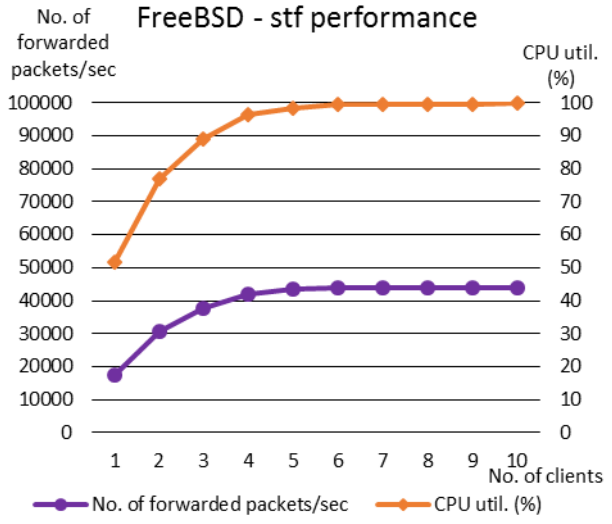

Fig. 4. FreeBSD stf forwarded packets and CPU utilization

Evaluation of the results:

Despite the fact that packet loss occurred in all cases, the proportion of it was always very low and it increased with more clients. (The maximum value of it was 0.061% with ten clients, which means about 6 packets from 10.000 packets were lost.)

The average, the standard deviation and the maximum value of the response times were increasing with higher load on the 6to4 relay computer, but the average value did not exceed 1.63 milliseconds with ten clients.

The CPU utilization were increasing continuously, but not linearly.

The deviation of the CPU utilization were higher with 4, 5, 6 and 7 clients, which indicates some fluctuation in the utilization.

The memory consumption was almost constant and very low, and the maximum value of it was 11.14kB with ten clients.

The traffic volume increased until the system reached its limit with 9 clients. With 10 clients, the number of transferred packets were slightly decreased from 73129 to 73050.

A. *OpenWRT (Attitude Adjustment) 12.09_x86 – sit*

The results have been listed in Table II, whereas the graphical representation of the forwarded packets per second and the CPU utilization are shown in Fig. 3.

Evaluation of the results:

The packet loss ratio was always very low and it strictly increased with the number of clients. The maximum value of it was 0.089% with ten clients.

The average and the standard deviation value of the response times were increasing with higher load on the 6to4 relay computer, but the average value did not exceed 2.16 milliseconds with ten clients.

The CPU utilization with two clients was 4.5 times greater than the value of one client. Then the slope was reduced, until the CPU approached its maximum capacity with 6 clients.

The standard deviation of the CPU utilization were under 10% in each case, which indicates consistent utilization of the CPU.

The memory consumption was almost constant and very low, and the maximum value of it was 10.75kB with seven clients.

The traffic volume increased until the system reached its limit with 9 clients. With 10 clients, the number of transferred packets were decreased by 0.97% from 59332 to 58763.

B. *FreeBSD 9.1_x86 – stf*

The results have been listed in Table III, whereas the graphical representation of the forwarded packets per second and the CPU utilization are shown in Fig. 4.

Evaluation of the results:

The packet loss ratio was always very low and starting from two clients it increased with the number of clients, whereas the value of it was the same with one and five clients. The maximum value of it was 0.019% with ten clients.

The average and the standard deviation value of the response times were increasing with higher load on the 6to4 relay computer, but the average value did not exceed 3.13

milliseconds with ten clients. The maximum value of the response times showed some fluctuation

One client could generate 51.53% load on the CPU. The CPU utilization was increasing continuously, but not linearly, until the CPU reached its almost maximum capacity (99.44%) with 6 clients.

The standard deviation of the CPU utilization was under 10% in each case, whereas it was very small (0.46%) with ten clients. This phenomenon indicates consistent utilization of the CPU.

The memory consumption was extremely low and it was growing almost continuously.

The traffic volume increased until the system reached its limit with 6 clients. From this point the throughput of the system started very slightly fluctuating. The maximum value of the number of transferred packets per second was 43970 with 9 clients.

VII. Comparison of the Results

All of the tested implementations proved to be reliable and the packet loss ratios of the different implementations were always very low. The packet loss ratio of the Linux and OpenWrt implementations seriously increased with the number of clients, whereas the packet loss of FreeBSD stf did not show much increase in the function of the load.

Under high load conditions, Linux sit tunnel forwarded the most packets per second and OpenWrt sit was the second one. The FreeBSD system was the last competitor in the performance comparison. At 10 clients, Linux outperformed FreeBSD by 1.67 times.

All of the implementations use negligibly small amount of memory, which is usually proportional to the generated load.

With one client, all of the implementations forwarded similar number of packets, but with significantly different CPU utilization, which property can explain the high degree of difference in the performance with more clients. Linux sit 6to4 relay implementation used 1.76% of CPU with one client, whereas FreeBSD stf used 51.53%, which means about 29 times difference.

VIII. Conclusion

The 6to4 protocol is a useful transition technique in a situation, where two IPv6 enabled hosts have to communicate over an IPv4 only network. All of the tested open source 6to4 relay implementations are viable solutions in production networks, but Linux sit showed the best performance characteristics, whereas OpenWrt sit was the second best one. In an environment, where a BSD system is preferred, FreeBSD stf is a usable solution as well.

The authors believe that their work has contributed to the early adoption of the IPv6 protocol and the published results and methodology are valuable for both researchers and network professionals.

Acknowledgement

The measurement of the performances of the different 6to4 implementations was the BSc thesis (final project) work of the second author at the Dept. of Telecommunications, Széchenyi István University under the supervision of the third author.

References

[1] S. Bradner and A. Mankin, “The recommendation for the IP next generation protocol,” IETF, January 1995. (RFC 1752)

[2] M. Waiser, “Whatever happened to the Next-Generation Internet?,” *Communications of the ACM*, vol. 44, pp. 61-69, Sep. 2001.

[3] S. Deering and R. Hinden, “Internet protocol, version 6 (IPv6) specification,” IETF, December 1998. (RFC 2460)

[4] Google, “IPv6 statistics”, http://www.google.com/ipv6/statistics.html

[5] G. Huston, “IPv4 address report,” Available: http://www.potaroo.net/tools/ipv4/index.html

[6] L. Smith and I. Lipner, “Free pool of IPv4 address space depleted,” Number Resource Organization, February 2011. Available: https://www.nro.net/news/ipv4-free-pool-depleted

[7] E. Nordmark and R. Gilligan, “Basic transition mechanisms for IPv6 hosts and routers,” IETF, October 2005. (RFC 4213)

[8] M. Bagnulo, A Sullivan, P. Matthews and I. Beijnum, “DNS64: DNS extensions for network address translation from IPv6 clients to IPv4 servers,” IETF, April 2011. ISSN: 2070-1721 (RFC 6147)

[9] M. Bagnulo, P. Matthews and I. Beijnum, “Stateful NAT64: network address and protocol translation from IPv6 clients to IPv4 servers,” IETF, April 2011. ISSN: 2070-1721 (RFC 6146)

[10] G. Lencse and S. Répás, “Performance analysis and comparison of different DNS64 implementations for Linux, OpenBSD and FreeBSD,” in *Proc. 27th IEEE Int. Conf. on Advanced Information Networking and Applications (AINA-2013)*, Barcelona, 2013, pp. 877-884. DOI: 10.1109/AINA.2013.80

[11] G. Lencse and S. Répás, "Performance analysis and comparison of the TAYGA and of the PF NAT64 implementations," in *Proc. 36th Int. Conf. on Telecommunications and Signal Processing (TSP-2013)*, Rome, 2013, pp. 71-76. DOI: 10.1109/TSP.2013.6613894

[12] S. Répás, T. Hajas and G. Lencse, “Application compatibility of the NAT64 IPv6 transition technology,” in *Proc. 37th Int. Conf. on Telecomm. and Signal Proc. (TSP-2014)*, Berlin, 2014, pp. 49-55.

[13] A. Conta and S. Deering, “Generic packet tunneling in IPv6 specification,” IETF, December 1998. (RFC 2473)

[14] SixXS - IPv6 Deployment & Tunnel Broker, https://www.sixxs.net/main/

[15] Hurricane Electric Free IPv6 Tunnel Broker, https://tunnelbroker.net/

[16] R. Despres, “IPv6 rapid deployment on IPv4 infrastructures (6rd),” IETF, January 2010. ISSN: 2070-1721 (RFC 5569)

[17] C. Huitema, “Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs),” IETF, February 2006. (RFC 4380)

[18] F. Templin, T. Gleeson and D. Thaler, “Intra-Site Automatic Tunnel Addressing Protocol (ISATAP),” IETF, March 2008. (RFC 5214)

[19] B. Carpenter and K. Moore, “Connection of IPv6 domains via IPv4 clouds,” IETF, February 2001. (RFC 3056)

[20] P. Wu, Y. Cui, J. Wu, J. Liu, and C. Metz, “Transition from IPv4 to IPv6: A state-of-the-art survey,” *IEEE Communications Surveys & Tutorials*, vol. 15, pp. 1407-1424, Jul. 2013.

[21] R. Gilligan and E. Nordmark, “Transition mechanisms for IPv6 hosts and routers,” IETF, August 2000. (RFC 2893)

[22] M. Cotton, L. Vegoda, R. Bonica and B. Haberman, “Special-purpose IP address registries,” IETF, April 2013. ISSN: 2070-1721 (RFC 6890)

[23] C. Partridge, T. Mendez and W. Milliken, “Host anycasting service,” IETF, November 1993. (RFC 1546)

[24] C. Huitema, “An anycast prefix for 6to4 relay routers,” IETF, June 2001. (RFC 3068)

[25] D. Malone, “Counting 6to4 relay routers,” *SIGCOMM Computer Communication Review*, vol. 36, pp. 79-82, Jan. 2006.

[26] RIPEstat, https://stat.ripe.net

[27] P. Savola and C. Patel, “Security considerations for 6to4,” IETF, December 2004. (RFC 3964)

[28] G. Lencse and S. Répás, “Performance analysis and comparison of 6to4 relay implementations,” *International Journal of Advanced Computer Science and Applications*, vol. 4, pp. 13-21, Sep. 2013. DOI: 10.14569/IJACSA.2013.040903

[29] Open Source Initiative, “The open source definition”, http://opensource.org/docs/osd

[30] Free Software Foundation, “The free software definition”, http://www.gnu.org/philosophy/free-sw.en.html